PROGRAMACIÓN ORIENTADA A OBJETOS

Blautzik - Gasior - Videla

Julio - Diciembre de 2021

Introducción

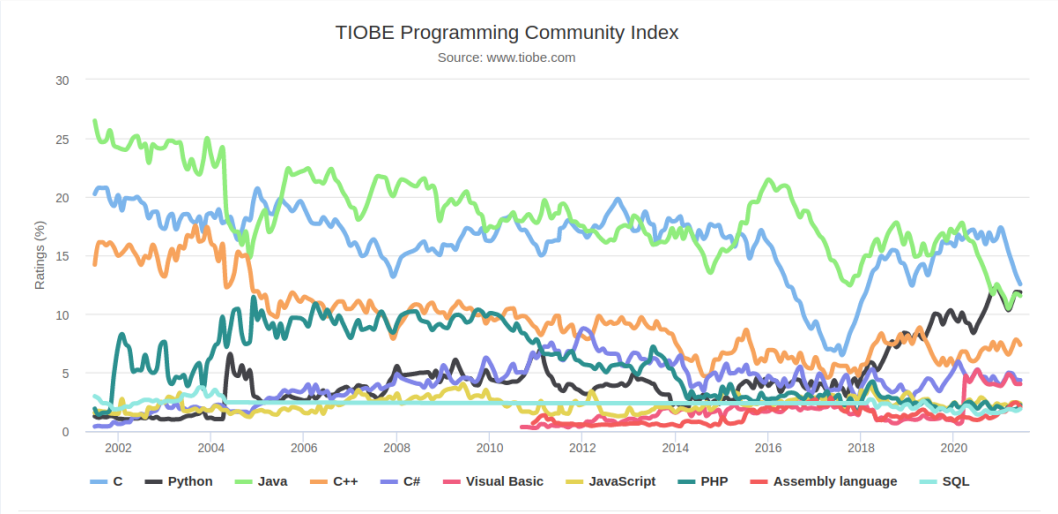
Definición:

Programación Orientada a Objetos: es el paradigma de programación en el cual los elementos de primer orden son los **objetos**.

En los **objetos**, los **datos** y los **métodos** que manipulan esos datos, se mantienen juntos como una unidad.

Los **datos** representan el estado interno del objeto. Los **métodos** son los que nos permiten manipular los **datos** de ese objeto.

Lenguajes de Programación Orientados a Objetos



Lenguajes de Programación Orientados a Objetos

Jul 2021	Jul 2020	Change	Programming Language	Ratings	Change
1	1		C	11.62%	-4.83%
2	2		Java	11.17%	-3.93%
3	3		Python	10.95%	+1.86%
4	4		C++	8.01%	+1.80%
5	5		C#	4.83%	-0.42%

¿Por qué Java?

1. Es uno de los lenguajes más utilizados en el mundo
2. Principalmente es un lenguaje Orientado a Objetos
3. Portátil (aunque esta característica deja de ser relevante)
4. Excelente desempeño y escalabilidad
5. Amplio soporte para la seguridad

Hola, mundo!

```java
public class App {
  public static void main(String[] args) {
    System.out.println("Hola, mundo!");
  }
}
```

Objetos

Definición:

Un objeto es una máquina de software que contiene datos y expone operaciones

“An object is a software machine allowing programs to access and modify a collection of data.”

[Meyer 2004]

Objetos (en otras palabras:)

Al término **‘objeto’** podemos traducirlo en nuestro lenguaje mediante la palabra **‘cosa’**.

No necesita existir en el mundo real, ser tangible, ni ser pequeño, o demasiado grande.

Estamos ante la posibilidad de modelar **‘objetos’** para poder utilizarlos como pequeñas **máquinas de software** que fundamentalmente:

- saben cosas (su estado interno).
- saben hacer cosas (con sus datos internos).

Solamente debe importarnos **qué cosas pueden hacer los objetos por nosotros**, y dejamos de lado (por el momento o definitivamente, según el caso) el conocimiento de las cosas que **‘saben’** los objetos.

Algunos objetos

CuentaGanado

Figure 1: Un Cuentaganado

- Características (atributos): contador.
- Sabe hacer (métodos): contar, mostrar y reiniciarse.

Algunos objetos

Circulo

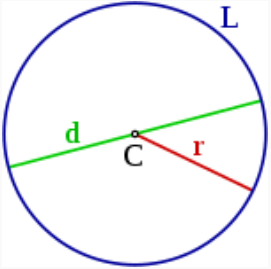


Figure 2: Un círculo

- Características (atributos): radio.
- Sabe hacer (métodos): obtener su radio, su diámetro, su área y su perímetro.

Algunos objetos

Perro

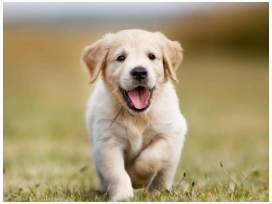


Figure 3: Un perro

- Características (atributos): nombre, raza, edad, color, etc.
- Sabe hacer (métodos): ladrar, comer, dormir, mover la cola, etc.

Algunos objetos

Automóvil

Figure 4: Un automóvil

- Características (atributos): Marca, modelo, año, cantidad de puertas, estado, etc.
- Sabe hacer (métodos): arrancar, acelerar, frenar, girar, etc.

Miembros

Lo que hace a un objeto, su esencia práctica, es que **podemos manipularlo a través de un grupo de operaciones bien definidas a las cuales llamamos miembros**. Los miembros de un objeto se invocan por medio de mensajes, y suele ser común confundir mensajes con miembros. Haremos una tosca pero práctica distinción:

- **Miembro** es cada una de las posibilidades de manipulación que nos proporciona un objeto. Se evidencia en tiempo de codificación.
- **Mensaje** es la invocación efectiva de esos miembros. Se evidencia en tiempo de ejecución.

Mensajes

La semántica introducida es el componente que cambia las reglas del juego.

En programación estructurada hacíamos esto :

- **f(o,x)** estamos diciendo “realizar la función f con o y x”.

Mediante la programación orientada a objetos, lo convertiremos en esto:

- **o.f(x)** estamos diciendo “objeto o, hacé f con x”.

Y aquí hemos cambiado un paradigma.

Mensajes

Con la programación orientada a objetos pasamos a depender de mensajes que nos derivan a los objetos adecuados sin necesidad de ejercer control directo.

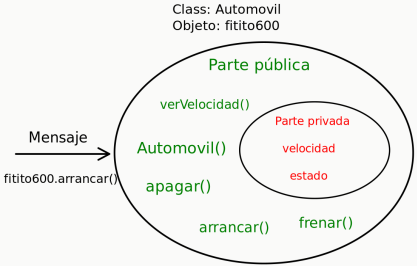


Figure 5: Paso de mensajes

Perdemos el control de quién lo va a interpretar. Sólo podemos esperar que el receptor reaccione apropiadamente.

Mensajes

Ni el que envía depende del que recibe, ni viceversa.

La acción en la programación orientada a objetos se inicia por la transmisión de un **mensaje** a un agente (un objeto) responsable por las acciones.

El mensaje codifica el pedido de una acción y se acompaña por cualquier información adicional (argumentos/parámetros) necesaria para llevar adelante el pedido.

El **receptor** es el objeto al cual el **mensaje** está dirigido.

Si éste acepta el **mensaje**, está aceptando la **responsabilidad** de llevar adelante la acción indicada.

En respuesta a un mensaje, el **receptor** llevará a cabo un método para satisfacer el pedido.

Consultas

Los miembros que nos permiten **obtener propiedades** de un objeto se denominan consultas. No deben develar como están implementadas, ya sea por algún cálculo o por almacenamiento.

```
class Persona {
  Integer calcularEdad() { ... }
  Date getFechaNacimiento() { ... }
}
```

Nos damos cuenta que traicionamos el ocultamiento de información: se sabe a ciencia cierta que lo que se almacena es la fecha de nacimiento, y que la edad se calcula.

Mensajes

Consultas

Respetando el principio de acceso uniforme, deberíamos hacerlo de la siguiente manera:

```
class Persona {
  Integer getEdad() { ... }
  Date getFechaNacimiento() { ... }
}
```

Para devolver los valores las consultas pueden requerir de ciertos parámetros adicionales. Lo importante es que las consultas **no alteran el estado de los objetos**, por lo que eso las define.

Mensajes

Comandos

Los comandos son peticiones de acción hacia un objeto. Son la contrapartida de las consultas ya que se caracterizan por modificar el estado de los objetos. Un ejemplo podría ser el siguiente:

```
class Automovil {
  void apagarLuces() { ... }
  void encenderLuces() { ... }
}
```

Como podemos ver, es evidente que se cambiará el estado interno del Auto, aunque la consecutiva invocación puede dejar de tener sentido: no sirve apagar una luz apagada.

Clases

Imaginemos que una clase es una plantilla, un molde, que se usa para crear un nuevo objeto de esa **clase**.

Las clases tienen tres características:

- Nombre.
- Atributos
- Operaciones o métodos.

Ese conjunto de características son las que definen el tipo de entidad del que estamos hablando. Distinguimos a los individuos por esas características y sabemos a que colectivo pertenece.

Esta distinción entre individuos y colectivo es la que debemos hacer al pensar entre objetos y clases.

Clases

Cada individuo de determinada clase dará valores a esos atributos y comportamiento efectivo a esas operaciones. Definir la clase Estudiante:

```
public class Estudiante {
  // atributos, constructor, métodos
}
```

Nos permite escribir:

```
Estudiante unEstudiante = new Estudiante("Juan");
```

En esa simple sentencia tenemos la relación que existe entre un objeto y una clase. “Un estudiante es una nueva instancia de Estudiante, en este caso su nombre será Juan”.

Clases

En conclusión:

- Los objetos tienen identidad y un estado interno particular.
- La clase no tiene identidad, ya que es un molde para todos los objetos, no tiene valores sino “espacios” preparados para adquirirlos.
- Si bien los objetos son los que reciben los mensajes y ejecutan acciones, las clases imponen qué instrucciones deberán ejecutarse y de qué modo en cada caso.

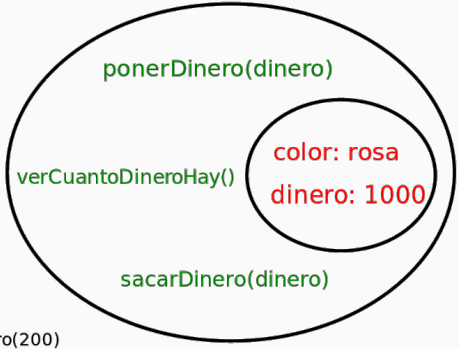
Un ejemplo práctico: La class Monedero

Defina una clase ‘Monedero’ que permita gestionar la cantidad de dinero de que una persona dispone en un momento dado. La clase deberá tener un constructor que permitirá crear un monedero con una cantidad de dinero inicial y deberá definir un método para meter dinero en el monedero, otro para sacarlo y finalmente, otro para consultar el disponible; solo podrá conocerse la cantidad de dinero del monedero a través de este último método. Por supuesto, no se podrá sacar más dinero del que haya en un momento dado en el monedero. Para probar el funcionamiento de la clase, escriba un método ‘main’ con una serie de instrucciones que hagan uso de los métodos definidos.

Un ejemplo práctico: La class Monedero

class Monedero
objeto: miMonederoRosa

miMonederoRosa



Mensajes:
miMonederoRosa.ponerDinero(200)
miMonederoRosa.sacarDinero(150)
miMonederoRosa.verCuantoDineroHay()
retorna 1050

Figure 6: Modelamos un monedero

Un ejemplo práctico: La class Monedero

```
public class Monedero{
    private double dinero;
    public Monedero(double dineroInicial){
        dinero = dineroInicial;
    }
    public double sacarDinero(double cantidadAsacar){
        if(dinero >= cantidadAsacar){
            dinero -=cantidadAsacar:
            return cantidadAsacar;
        }
        else
            reeturn 0;
    }
```

Un ejemplo práctico: La class Monedero

```
public void ponerDinero(double dineroAingresar) {
    dinero += dineroAingresar;
}
public double verCuantoDineroHay() {
    return dinero;
}
```

Un ejemplo práctico: La class Monedero

Veamos al Monedero en acción en eclipse...